Data Mining Final Project Report

By Vanessa Arteaga Gonzalez, Kayla Laufer, Jonathan Ramos, and Luisa Rosa

Introduction

There were many routes we could have taken when choosing a course project. Our initial plan was to enter a Kaggle competition; after some research, we settled on a competition called BirdCLEF 2024 — a competition that revolves around using machine learning techniques to identify under-studied Indian bird species by sound. After a more in-depth analysis of the project, however, we decided that the quantity of data needed (23.43 GB) was too large for our individual computers and for the time remaining before the project deadline. We could not find another Kaggle competition or project idea that fit our timeline. Thus, we ended up choosing the default project provided in class.

In this project, we use classification algorithms, aiming to understand their efficacy on a real-world dataset of census income data. Our objective is to meticulously analyze and document the behavior of various classification algorithms, shedding light on their performance metrics and capabilities. The dataset encompasses a diverse array of individual attributes ranging from educational backgrounds to demographic and family information. Each data instance contains crucial features pertinent to an individual's socio-economic profile. The main task of our project is to predict whether an individual's annual income surpasses $50,000. Our approach involves the utilization of two primary files: census-income.data for training our classifier and census-income.test for evaluating its performance. We meticulously document the experimental results for each method used, including accuracy, false positive rate, false negative rate, and other pertinent statistics. This project aims to explore classification algorithms within the context of income prediction comprehensively. By rigorously assessing algorithmic performance and methodology, we aim to contribute valuable insights to the realm of machine learning and data analysis.

Methodology

For this project, we decided to code in Python, a language we are all familiar with. We decided to use Google Collab as our IDE, which allowed all of the members in the group to work on and edit a single program. Of the classification methods covered in class, we tested KNN (with and without bagging), Decision Tree, Random Forest (with and without bagging), Naive Bayes (with and without Laplace smoothing), Logistic Regression, and an ensemble classifier to see which gave the highest accuracy. Methods such as K-means, K-medoids, and Hierarchical clustering were not used, as these clustering methods are primarily used for unsupervised learning. For unsupervised learning clustering, training data (observations, measurements, etc.) are not accompanied by target values or labels. The default training dataset includes labels, making it suitable for supervised learning and, thus, unsuitable for unsupervised learning methods.

Preparing the Data

To ensure the integrity and completeness of our dataset, we employed a meticulous data-cleaning process. Using the K-nearest neighbors (KNN) imputation method with a k-value of 5, we replaced missing numerical values, thereby preserving the statistical characteristics of the dataset. Additionally, categorical missing values were imputed using the mode of their respective columns. Acknowledging the importance of addressing class imbalance, we utilized Python's Synthetic Minority Oversampling Technique (SMOTE) to rectify any imbalance issues within the dataset. SMOTE generated synthetic samples of the minority class, thus mitigating the imbalance and ensuring a more representative training dataset for our classification models. By meticulously cleaning and balancing the dataset, we aimed to create a robust and reliable foundation for our subsequent analyses and model training processes. We first ran all tested algorithms with the unprocessed dataset to get a general overview of which performed better. Of all initial classifiers, Random Forest performed the best, a trend that continued into the prepared data.

Random Forest

The algorithm with the best performance for this data set was Random Forest. This ensemble learning method constructs multiple decision trees during the training process and outputs the mode of the classes as a prediction. When running Random Forest with imbalanced data and no feature selection, we got a high accuracy of 85.1422%, signifying that the algorithm performed well. We also calculated the following metrics:

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 84.6631% F1-score: 0.6595 Precision: 0.6934 Confusion Matrix: [[11366 1069] [1428 2418]] TP: 2418 FP: 1069 FN: 1428 TN: 11366 Recall: 0.6287 Specificity (True Negative Rate): 0.9140 False Positive Rate (FPR): 0.0860 False Negative Rate (FNR): 0.3713	Accuracy: 79.9030% F1-score: 0.5634 Precision: 0.5787 Confusion Matrix: [[10898 1537] [1735 2111]] TP: 2111 FP: 1537 FN: 1735 TN: 10898 Recall: 0.5489 Specificity (True Negative Rate): 0.8764 False Positive Rate (FPR): 0.1236 False Negative Rate (FNR): 0.4511	Accuracy: 85.1422% F1-score: 0.6577 Precision: 0.7215 Confusion Matrix: [[11538 897] [1522 2324]] TP: 2324 FP: 897 FN: 1522 TN: 11538 Recall: 0.6043 Specificity (True Negative Rate): 0.9279 False Positive Rate (FPR): 0.0721 False Negative Rate (FNR): 0.3957	Accuracy: 83.4347% F1-score: 0.6225 Precision: 0.6741 Confusion Matrix: [[11360 1075] [1622 2224]] TP: 2224 FP: 1075 FN: 1622 TN: 11360 Recall: 0.5783 Specificity (True Negative Rate): 0.9136 False Positive Rate (FPR): 0.0864 False Negative Rate (FNR): 0.4217

The Random Forest algorithm produced a high precision value and F1-score, further showing that this classifier was able to classify a large majority of the data correctly. The high specificity

value, coupled with a low FPR and FNR, suggests that the classifier was largely accurate in identifying the negative class.

The Random Forest model performs better in terms of accuracy when the dataset is imbalanced but tends to have better recall when the dataset is balanced. Feature selection generally leads to a reduction in performance, suggesting the need for careful selection of features or reconsideration of the feature selection method used. This analysis highlights the importance of considering both data balance and feature selection in the context of model performance and the trade-offs between different performance metrics.

We also ran Bagging with Random Forest to see if it significantly improved the accuracy of the classifier. This technique involves training multiple instances of a classifier and combining their predictions. The accuracy did go up by some small fraction, landing at 85.7318%. Other metrics were also marginally improved:

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 85.1913% F1-score: 0.6753 Precision: 0.7005 Confusion Matrix: [[11363 1072] [1339 2507]] TP: 2507 FP: 1072 FN: 1339 TN: 11363 Recall: 0.6518 Specificity (True Negative Rate): 0.9138 False Positive Rate (FPR): 0.0862 False Negative Rate (FNR): 0.3482	Accuracy: 80.6523% F1-score: 0.5856 Precision: 0.5927 Confusion Matrix: [[10905 1530] [1620 2226]] TP: 2226 FP: 1530 FN: 1620 TN: 10905 Recall: 0.5788 Specificity (True Negative Rate): 0.8770 False Positive Rate (FPR): 0.1230 False Negative Rate (FNR): 0.4212	Accuracy: 85.7318% F1-score: 0.6694 Precision: 0.7394 Confusion Matrix: [[11606 829] [1494 2352]] TP: 2352 FP: 829 FN: 1494 TN: 11606 Recall: 0.6115 Specificity (True Negative Rate): 0.9333 False Positive Rate (FPR): 0.0667 False Negative Rate (FNR): 0.3885	Accuracy: 83.9629% F1-score: 0.6328 Precision: 0.6891 Confusion Matrix: [[11420 1015] [1596 2250]] TP: 2250 FP: 1015 FN: 1596 TN: 11420 Recall: 0.5850 Specificity (True Negative Rate): 0.9184 False Positive Rate (FPR): 0.0816 False Negative Rate (FNR): 0.4150

While the precision, F1-score, and specificity all improved the performance of Random Forest without Bagging, this improvement is minimal.

The Bagging with Random Forest model tends to perform best without feature selection, regardless of whether the data is balanced or not. This suggests that the model benefits from having access to all available features, likely because Random Forest can inherently manage irrelevant features through its ensemble approach. Balancing improves recall but slightly reduces specificity and overall accuracy, indicating a trade-off that might be acceptable or even desirable depending on the specific application needs (e.g., when it's critical to capture as many positives as possible). Feature selection appears to consistently lower performance metrics, implying that the method or criteria for feature selection may need adjustment or that the model's nature (ensemble and tree-based) already optimally handles feature relevance internally.

It is important to note that the Random Forest algorithm performed better with an unbalanced data set and without feature selection. While the algorithm performs only marginally worse with a balanced data set, implementing feature selection has a more detrimental effect on the model's accuracy. This could be due to the fact that when feature selection is introduced, it removes important information that could have been used by Random Forest to capture the underlying patterns in the data. In this case, it would decrease model performance, especially in the context of unbalanced data, where each feature might carry valuable predictive information.

K-Nearest Neighbors

The K-Nearest Neighbors algorithm (KNN) is a non-parametric method that consists of predicting a classifier based on its nearest neighbors. This algorithm was run using k = [3, 10,15, 20, 25, 30]. When running the classifier with imbalanced data, with feature selection, and without bagging, it performed best with k = 20, scoring 84.58% accuracy. The following metrics were also calculated:

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 66.1077% F1-score: 0.4127 Precision: 0.3494 Confusion Matrix: [[8824 3611] [1907 1939]] TP: 1939 FP: 3611 FN: 1907 TN: 8824 Recall: 0.5042 Specificity (True Negative Rate): 0.7096 False Positive Rate (FPR): 0.2904 False Negative Rate (FNR): 0.4958	Accuracy: 78.0787% F1-score: 0.5908 Precision: 0.5284 Confusion Matrix: [[10136 2299] [1270 2576]] TP: 2576 FP: 2299 FN: 1270 TN: 10136 Recall: 0.6698 Specificity (True Negative Rate): 0.8151 False Positive Rate (FPR): 0.1849 False Negative Rate (FNR): 0.3302	Accuracy: 80.2961% F1-score: 0.3328 Precision: 0.8316 Confusion Matrix: [[12273 162] [3046 800]] TP: 800 FP: 162 FN: 3046 TN: 12273 Recall: 0.2080 Specificity (True Negative Rate): 0.9870 False Positive Rate (FPR): 0.0130 False Negative Rate (FNR): 0.7920	Accuracy: 84.5833% F1-score: 0.6211 Precision: 0.7405 Confusion Matrix: [[11714 721] [1789 2057]] TP: 2057 FP: 721 FN: 1789 TN: 11714 Recall: 0.5348 Specificity (True Negative Rate): 0.9420 False Positive Rate (FPR): 0.0580 False Negative Rate (FNR): 0.4652

The KNN algorithm scores well in terms of accuracy, precision, and specificity with imbalance data but is less than optimal on F1-score and recall, which, coupled with a moderate false negative rate, implies that the classifier sometimes misclassifies elements in the positive class as negative. This leads to a slightly less optimal performance when compared to Random Forest.

KNN's performance varies widely based on the balance of the dataset and whether feature selection is applied. Feature selection consistently improves model metrics in balanced and imbalanced datasets, indicating that reducing dimensionality helps KNN focus on the most informative features. However, the model's effectiveness in scenarios without feature selection suggests that KNN may struggle with high-dimensional data or when irrelevant features are

included. These insights highlight the importance of appropriate preprocessing and feature engineering when using KNN for classification tasks.

When combining KNN with a bagging technique, we used a k-value of 25. However, there was nearly no difference in metrics:

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 64.2590% F1-score: 0.4105 Precision: 0.3363 Confusion Matrix: [[8436 3999] [1820 2026]] TP: 2026 FP: 3999 FN: 1820 TN: 8436 Recall: 0.5268 Specificity (True Negative Rate): 0.6784 False Positive Rate (FPR): 0.3216 False Negative Rate (FNR): 0.4732	Accuracy: 77.2066% F1-score: 0.5919 Precision: 0.5129 Confusion Matrix: [[9879 2556] [1155 2691]] TP: 2691 FP: 2556 FN: 1155 TN: 9879 Recall: 0.6997 Specificity (True Negative Rate): 0.7945 False Positive Rate (FPR): 0.2055 False Negative Rate (FNR): 0.3003	Accuracy: 80.3145% F1-score: 0.3371 Precision: 0.8241 Confusion Matrix: [[12261 174] [3031 815]] TP: 815 FP: 174 FN: 3031 TN: 12261 Recall: 0.2119 Specificity (True Negative Rate): 0.9860 False Positive Rate (FPR): 0.0140 False Negative Rate (FNR): 0.7881	Accuracy: 84.3683% F1-score: 0.6251 Precision: 0.7210 Confusion Matrix: [[11614 821] [1724 2122]] TP: 2122 FP: 821 FN: 1724 TN: 11614 Recall: 0.5517 Specificity (True Negative Rate): 0.9340 False Positive Rate (FPR): 0.0660 False Negative Rate (FNR): 0.4483

The use of Bagging with KNN demonstrates varying effectiveness, heavily influenced by the balance of the dataset and the application of feature selection. While feature selection consistently improves accuracy and the balance between precision and recall under both balanced and imbalanced conditions, the impact of balancing the data on performance appears more nuanced. Balancing typically enhances recall, suggesting improved detection of the minority class, which is crucial for applications requiring high sensitivity. However, KNN's overall performance shows moderate accuracy and precision with high specificity, indicating a strong ability to identify the negative class but potential challenges in correctly classifying the positive class. This analysis underscores the value of feature selection in improving the performance of KNN, particularly in ensemble settings where reducing noise and focusing on relevant features can significantly enhance the model's predictive capabilities

Compared to the Random Forest model, KNN significantly benefits from feature selection, suggesting that reducing feature space complexity is particularly effective for KNN in ensemble settings. This analysis underscores the critical role of feature selection in enhancing KNN's predictive capabilities, especially in scenarios where noise reduction and focus on relevant features are paramount.

Logistic Regression

Logistic Regression is a linear model used for binary classification by modeling the probability of each possible outcome. This algorithm scored decently with the following metrics:

Balance w/o Feature Selection	Balance with Feature Selection ** Max iteration for feature selection convergence was reached **	Imbalance w/o Feature Selection	Imbalance with Feature Selection ** Max iteration for feature selection convergence was reached **
Accuracy: 64.7012% F1-score: 0.4188 Precision: 0.3427 Confusion Matrix: [[8463 3972] [1775 2071]] TP: 2071 FP: 3972 FN: 1775 TN: 8463 Recall: 0.5385 Specificity (True Negative Rate): 0.6806 False Positive Rate (FPR): 0.3194 False Negative Rate (FNR): 0.4615	Accuracy: 77.7041% F1-score: 0.6028 Precision: 0.5204 Confusion Matrix: [[9896 2539] [1091 2755]] TP: 2755 FP: 2539 FN: 1091 TN: 9896 Recall: 0.7163 Specificity (True Negative Rate): 0.7958 False Positive Rate (FPR): 0.2042 False Negative Rate (FNR): 0.2837	Accuracy: 79.7801% F1-score: 0.3779 Precision: 0.6916 Confusion Matrix: [[11989 446] [2846 1000]] TP: 1000 FP: 446 FN: 2846 TN: 11989 Recall: 0.2600 Specificity (True Negative Rate): 0.9641 False Positive Rate (FPR): 0.0359 False Negative Rate (FNR): 0.7400	Accuracy: 82.3537% F1-score: 0.5526 Precision: 0.6889 Confusion Matrix: [[11634 801] [2072 1774]] TP: 1774 FP: 801 FN: 2072 TN: 11634 Recall: 0.4613 Specificity (True Negative Rate): 0.9356 False Positive Rate (FPR): 0.0644 False Negative Rate (FNR): 0.5387

The accuracy, precision, and F1-score are decent for Logistic Regression, but fail to reach the performance levels of the Random Forest algorithm. Much like with the KNN classifier, this algorithm had difficulties correctly predicting positive values, as shown by the False Negative Rate. A high specificity, however, indicates that the Logistic Regression model was largely successful in handling the negative class.

This model worked best with feature selection. Logistic Regression's performance is markedly enhanced by feature selection, particularly in balanced datasets where it helps achieve a better trade-off between detecting both classes. In imbalanced conditions, while the accuracy remains high, the model's ability to identify the positive class weakly suggests that further tuning or alternative approaches might be necessary to handle class imbalance effectively, such as using different class weights or more sophisticated resampling techniques. The convergence issues also indicate that more attention needs to be given to the optimization process to ensure robust model training.

Naive Bayes

Naive Bayes is a probabilistic classifier based on Bayes' Theorem. It is considered "naive" because it assumes features to be independent of each other. With imbalance data and feature selection, this algorithm was slightly more accurate than the Logistic Regression model, scoring an accuracy of 83.3855%.

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 79.4054% F1-score: 0.4141 Precision: 0.6313 Confusion Matrix: [[11743 692] [2661 1185]] TP: 1185 FP: 692 FN: 2661 TN: 11743 Recall: 0.3081 Specificity (True Negative Rate): 0.9444 False Positive Rate (FPR): 0.0556 False Negative Rate (FNR): 0.6919	Accuracy: 68.0425% F1-score: 0.5599 Precision: 0.4149 Confusion Matrix: [[7768 4667] [536 3310]] TP: 3310 FP: 4667 FN: 536 TN: 7768 Recall: 0.8606 Specificity (True Negative Rate): 0.6247 False Positive Rate (FPR): 0.3753 False Negative Rate (FNR): 0.1394	Accuracy: 79.5713% F1-score: 0.4136 Precision: 0.6424 Confusion Matrix: [[11782 653] [2673 1173]] TP: 1173 FP: 653 FN: 2673 TN: 11782 Recall: 0.3050 Specificity (True Negative Rate): 0.9475 False Positive Rate (FPR): 0.0525 False Negative Rate (FNR): 0.6950	Accuracy: 83.3855% F1-score: 0.5972 Precision: 0.6988 Confusion Matrix: [[11571 864] [1841 2005]] TP: 2005 FP: 864 FN: 1841 TN: 11571 Recall: 0.5213 Specificity (True Negative Rate): 0.9305 False Positive Rate (FPR): 0.0695 False Negative Rate (FNR): 0.4787

Naive Bayes performs variably across different dataset conditions, showing a general trend of higher specificity but lower recall. The application of feature selection can markedly improve its performance, particularly in recognizing the positive class in imbalanced datasets. However, this improvement in recall with feature selection often reduces precision and specificity, indicating a shift in the model's threshold for classifying positive cases. These insights suggest that while Naive Bayes can be effective, especially in scenarios favoring high specificity, careful consideration is needed in tuning and feature selection to balance its performance across different classes.

The Naive Bayes classifier was also run with Laplace Smoothing, a technique that handles zero-frequency issues by adding a constant to the frequency counts.

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 78.4903% F1-score: 0.3370 Precision: 0.6198 Confusion Matrix: [[11889 546] [2956 890]] TP: 890 FP: 546 FN: 2956 TN: 11889 Recall: 0.2314 Specificity (True Negative Rate): 0.9561 False Positive Rate (FPR): 0.0439	Accuracy: 71.4145% F1-score: 0.5675 Precision: 0.4416 Confusion Matrix: [[8574 3861] [793 3053]] TP: 3053 FP: 3861 FN: 793 TN: 8574 Recall: 0.7938 Specificity (True Negative Rate): 0.6895 False Positive Rate (FPR): 0.3105	Accuracy: 78.5148% F1-score: 0.3370 Precision: 0.6217 Confusion Matrix: [[11894 541] [2957 889]] TP: 889 FP: 541 FN: 2957 TN: 11894 Recall: 0.2311 Specificity (True Negative Rate): 0.9565 False Positive Rate (FPR): 0.0435	Accuracy: 78.2814% F1-score: 0.3153 Precision: 0.6176 Confusion Matrix: [[11931 504] [3032 814]] TP: 814 FP: 504 FN: 3032 TN: 11931 Recall: 0.2116 Specificity (True Negative Rate): 0.9595

False Negative Rate (FNR): 0.7686	False Negative Rate (FNR): 0.2062	False Negative Rate (FNR): 0.7689	False Positive Rate (FPR): 0.0405 False Negative Rate (FNR): 0.7884

Naive Bayes with Laplace smoothing performs consistently in terms of specificity across different conditions but struggles with recall. The application of feature selection can improve recall but often reduces overall accuracy and increases false negatives. This pattern suggests that while Naive Bayes is effective for datasets where positive classification is critical, it may require careful tuning and potentially additional strategies to improve its performance in identifying negative cases.

Decision Tree

Decision Trees are tree-based models that make decisions by splitting data into subsets, which are based on the value of features.

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 80.9287% F1-score: 0.6071 Precision: 0.5913 Confusion Matrix: [[10777 1658] [1447 2399]] TP: 2399 FP: 1658 FN: 1447 TN: 10777 Recall: 0.6238 Specificity (True Negative Rate): 0.8667 False Positive Rate (FPR): 0.1333 False Negative Rate (FNR): 0.3762	Accuracy: 78.6991% F1-score: 0.5256 Precision: 0.5546 Confusion Matrix: [[10892 1543] [1925 1921]] TP: 1921 FP: 1543 FN: 1925 TN: 10892 Recall: 0.4995 Specificity (True Negative Rate): 0.8759 False Positive Rate (FPR): 0.1241 False Negative Rate (FNR): 0.5005	Accuracy: 81.1007% F1-score: 0.6066 Precision: 0.5967 Confusion Matrix: [[10832 1603] [1474 2372]] TP: 2372 FP: 1603 FN: 1474 TN: 10832 Recall: 0.6167 Specificity (True Negative Rate): 0.8711 False Positive Rate (FPR): 0.1289 False Negative Rate (FNR): 0.3833	Accuracy: 82.2738% F1-score: 0.5938 Precision: 0.6473 Confusion Matrix: [[11286 1149] [1737 2109]] TP: 2109 FP: 1149 FN: 1737 TN: 11286 Recall: 0.5484 Specificity (True Negative Rate): 0.9076 False Positive Rate (FPR): 0.0924 False Negative Rate (FNR): 0.4516

In terms of accuracy, the Decision Tree model generally performs better than Logistic Regression, and Naive Bayes, but pales in comparison to the Random Forest model. It also scored lowest on specificity compared to the other models that were tested, suggesting the Decision Tree was the least effective at predicting the negative class.

The Decision Tree classifier exhibits a flexible performance across different dataset conditions, with its ability to handle imbalance and feature selection variably. While balancing generally helps in improving recall, feature selection seems to impact the model's sensitivity negatively by reducing its ability to identify true positives (as seen with lower recall with feature selection). This analysis suggests that while Decision Trees can be effective in various settings,

careful consideration needs to be given to feature selection and class distribution to optimize its performance. Adjusting the complexity of the tree (like setting the maximum depth) or using ensemble methods (like Random Forests) could also help improve robustness and accuracy.

Ensemble Classifier

After evaluating the metrics for each individual classifier, we tested an ensemble classifier — which combines the predictions of different models — on the data. This technique is meant to improve performance by combining the diverse perspectives of various models. The classifier included Random Forest, KNN, Logistic Regression, Naive Bayes with Laplace Smoothing, Decision Tree, and Bagging with Random Forest as base learners.

Balance w/o Feature Selection	Balance with Feature Selection	Imbalance w/o Feature Selection	Imbalance with Feature Selection
Accuracy: 84.8719% F1-score: 0.6444 Precision: 0.7244 Confusion Matrix: [[11586 849] [1614 2232]] TP: 2232 FP: 849 FN: 1614 TN: 11586 Recall: 0.5803 Specificity (True Negative Rate): 0.9317 False Positive Rate (FPR): 0.0683 False Negative Rate (FNR): 0.4197	Accuracy: 80.1056% F1-score: 0.5875 Precision: 0.5757 Confusion Matrix: [[10735 1700] [1539 2307]] TP: 2307 FP: 1700 FN: 1539 TN: 10735 Recall: 0.5998 Specificity (True Negative Rate): 0.8633 False Positive Rate (FPR): 0.1367 False Negative Rate (FNR): 0.4002	Accuracy: 85.2220% F1-score: 0.6108 Precision: 0.8082 Confusion Matrix: [[11987 448] [1958 1888]] TP: 1888 FP: 448 FN: 1958 TN: 11987 Recall: 0.4909 Specificity (True Negative Rate): 0.9640 False Positive Rate (FPR): 0.0360 False Negative Rate (FNR): 0.5091	Accuracy: 83.4777% F1-score: 0.5975 Precision: 0.7037 Confusion Matrix: [[11594 841] [1849 1997]] TP: 1997 FP: 841 FN: 1849 TN: 11594 Recall: 0.5192 Specificity (True Negative Rate): 0.9324 False Positive Rate (FPR): 0.0676 False Negative Rate (FNR): 0.4808

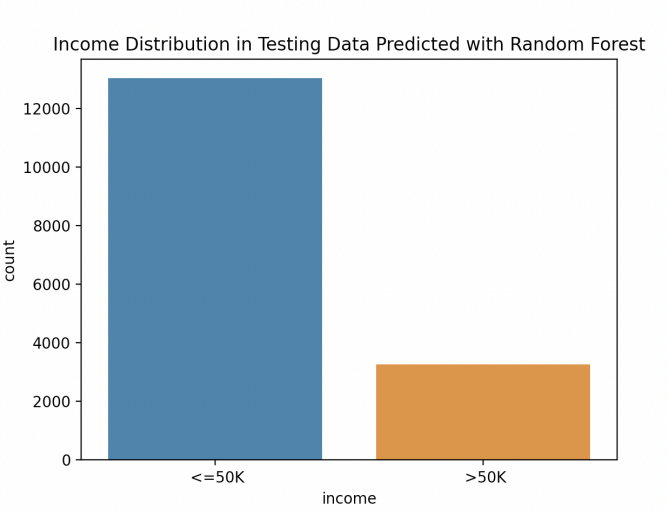
As shown by the above metrics, the ensemble classifier scores highly on accuracy, F1-score, and precision. However, it fails to achieve the best scores in these categories. While the high specificity indicates that the Ensemble classifier adequately recognizes the negative class, the FNR remains relatively high. Overall, the Ensemble classifier scores well, but fails to reach the same level as the Random Forest model.

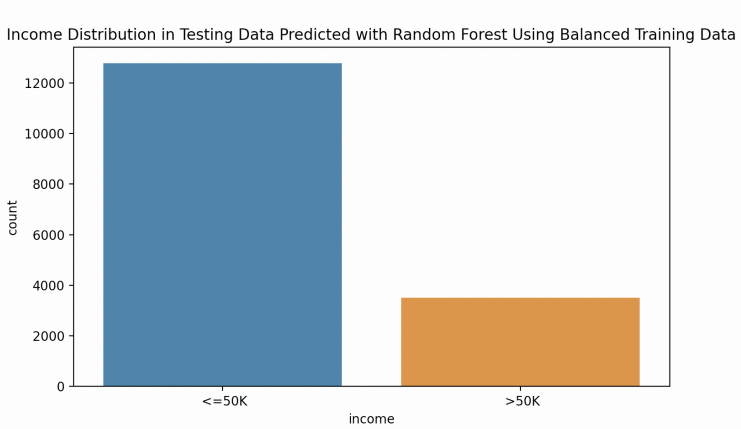
Data Visualization Tools

The sns.countplot function from the seaborn library generates a count plot that visualizes the distribution of the 'income' variable within the data dataset. Specifically, the plot displays 'income' categories along the x-axis, with two primary income brackets: "<= $50K" and "> $50K". The height of each bar represents the frequency of individuals within each income category. This visualization effectively illustrates the number of individuals falling into each

income bracket, providing a clear comparative view of the economic divisions captured in the dataset.

Income Distribution

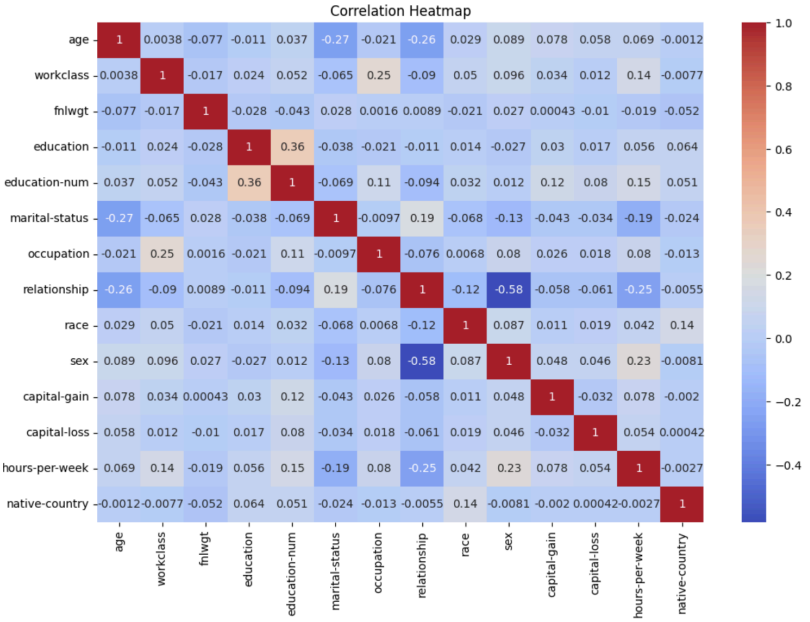




```
Income Distribution:

Training Data:
<=50K: 24,720 individuals
>50K: 7,841 individuals
Total: 24,720 + 7,841 = 32,561 individuals
Rate of <=50K: (24,720 / 32,561) * 100 ≈ 75.9%
Rate of >50K: (7,841 / 32,561) * 100 ≈ 24.1%

Testing Data (Predicted using Random Forest → best performing
algorithm):
<=50K: 13,023 individuals
>50K: 3,258 individuals
Total: 13,023 + 3,258 = 16,281 individuals
Rate of <=50K: (13,023 / 16,281) * 100 ≈ 79.9%
Rate of >50K: (3,258 / 16,281) * 100 ≈ 20.1%

Testing Data (Predicted using Random Forest with balanced data → best
performing algorithm):
<=50K: 13,023 individuals
>50K: 3,258 individuals
Total: 12,814 + 3,467 = 16,281 individuals
Rate of <=50K: (12,814 / 16,281) * 100 ≈ 78.7%
Rate of >50K: (3,467 / 16,281) * 100 ≈ 21.3%
```

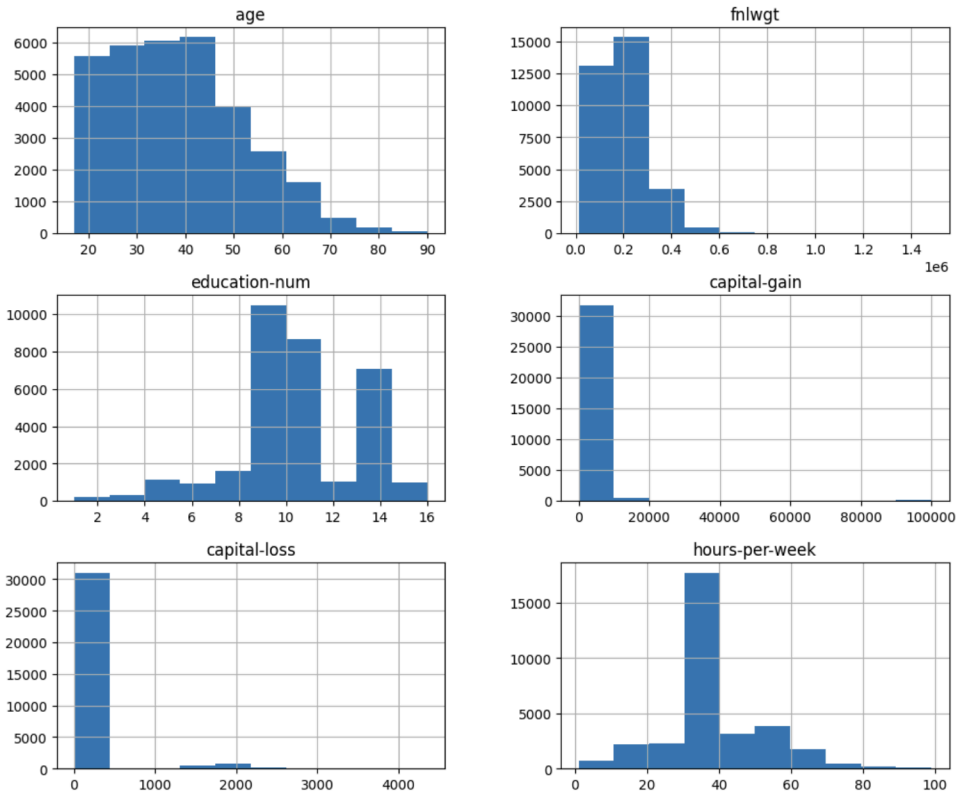
From the data, it's evident that incomes exceeding $50,000 are substantially less frequent than incomes at or below $50,000. The original training data shows significant imbalance with a ratio of about 3.15 to 1 (in favor of the <= $50K group). Despite the imbalance, the Random Forest model performs well, achieving high accuracy and resulting in a predicted ratio of 4 to 1 in the standard testing data. When using balanced training data, the model improves its prediction of the minority class (those earning > $50K), reducing the ratio to approximately 3.7 to 1. This data demonstrates the benefits of using a balanced dataset. Balancing the training data leads to more equitable predictions across different income groups, as evidenced by the closer ratio of income distribution in the predictions when using the balanced dataset. Such adjustments enhance the model's ability to accurately predict the minority class, resulting in a more representative and fair outcome. This underscores the importance of employing strategies like data balancing in training data mining models, especially when dealing with imbalanced datasets.

Heatmap



To visualize the correlations between numerical features, we start by importing the necessary libraries: seaborn (aliased as sns) for data visualization and matplotlib.pyplot (aliased as plt) for plot creation. We then prepare our data by removing the 'income' column, focusing solely on the remaining numerical features. A correlation matrix is calculated from these features and set as the data source for our heatmap. Using seaborn's heatmap function, we created a heatmap with annotations to display the exact correlation values. The color map 'coolwarm' is chosen to visually represent the strength and direction of the correlations—warm colors like red and orange indicate positive correlations, where variables increase together, while cool colors like blue suggest negative correlations, where one variable increases as the other decreases. Finally, we displayed the plot with plt.show(). It's worth noting that if you remove half of the heatmap along the diagonal marked by ones, you would not lose any information, as this represents redundant data mirroring across the diagonal.

Histogram plot



Each histogram chart represents a specific range of values. Each bar covers a range of numeric values called a bin, the height indicates the frequency of data points with a value within the corresponding bin. In the age histogram, the 39 to 47 age range has the highest values while over 40s was decreasing with age.

For the purposes of our project, the correlation analysis between the income column and the remaining columns of the dataset offers compelling insights. Income exhibits stronger correlations with “education-num”, “age”, “sex”, “capital gain”, and “hours per week”. Conversely, it negatively correlates with “marital status”, and “relationship”.

Future Work

Objective: The primary goal will be to enhance the model's ability to correctly identify positive cases (minority class) as positive (recall), which is crucial in scenarios where false negatives are more detrimental than false positives. For example, in medical testing, it is important to classify patients with certain illnesses so they are not undiagnosed.

Strategies to Improve Recall:

1. **Threshold Adjustment**: Adjusting the decision threshold of the classifier can shift the balance between sensitivity (recall) and specificity. By decreasing the threshold required to classify a positive instance, recall can be improved as the model becomes less conservative in predicting positives. This approach will likely reduce specificity as fewer negative predictions are made.
2. **Feature Engineering**: Developing or selecting features that are more predictive of the positive class could help improve recall. This might involve incorporating additional data sources or engineering domain-specific features that enhance the model's ability to discern positives accurately. We can also implement feature selection algorithms that maximize the mutual information between features and the positive class.
3. **Cost-sensitive Learning**: Implement a cost-sensitive learning framework where the cost of false negatives is set higher than false positives. This approach directly influences the learning algorithm to prioritize reducing false negatives over capturing all true negatives, thus improving recall.
4. **Ensemble Methods**: Utilizing ensemble methods like Random Forests or boosting can improve model robustness and recall. These methods aggregate predictions from multiple models, which can be tuned to favor recall by, for instance, majority voting mechanisms where a negative classification requires a higher consensus among individual classifiers. In our current version, we used Random Forest, Bagging, and Ensemble Learning. We can improve these models by adjusting parameters and ensuring base learners are independent.

Potential Trade-offs:

- **Impact on Specificity**: Any effort to improve recall must be carefully managed to assess the impact on specificity. In some applications, a significant drop in specificity might not be acceptable. It's important to evaluate the cost of missed negative cases versus the cost of false alarms.
- **Evaluation Metrics**: Besides tracking changes in specificity and recall, overall performance metrics like the F1-score, Precision, Confusion Matrix, etc, should also be monitored to ensure that overall model performance remains balanced.

Testing and Validation:

- **Iterative Testing**: Implement iterative testing phases where adjustments to improve recall are incrementally tested and validated to monitor their impact on other performance metrics.
- **Domain-Specific Validation**: Engage domain experts to validate if increases in recall yield practical benefits in the intended application area.

Works Cited

Chawla, N. V., Bowyer, K. W., Hall, L. O., & Kegelmeyer, W. P. (2002). SMOTE: synthetic minority over-sampling technique. *Journal of Artificial Intelligence Research*, *16*, 321–357.

Cox, D. R. (1958). The regression analysis of binary sequences. *Journal of the Royal Statistical Society: Series B (Methodological)*, *20*(2), 215–232.

Ho, T. K. (1995). Random decision forests. In *Proceedings of 3rd international conference on document analysis and recognition* (Vol. 1, pp. 278–282)

Wu, X., Kumar, V., Quinlan, J. R., Ghosh, J., Yang, Q., Motoda, H., … others. (2008). Top 10 algorithms in data mining. *Knowledge and Information Systems*, *14*(1), 1–37.

Breiman, L. (1996). Bagging predictors. *Machine Learning*, *24*(2), 123–140.

Pedregosa, F., Varoquaux, Ga"el, Gramfort, A., Michel, V., Thirion, B., Grisel, O., … others. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, *12*(Oct), 2825–2830.

Waskom, M., Botvinnik, Olga, O'Kane, Drew, Hobson, Paul, Lukauskas, Saulius, Gemperline, David C, … Qalieh, Adel. (2017). *mwaskom/seaborn: v0.8.1 (September 2017)*. Zenodo. https://doi.org/10.5281/zenodo.883859

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. *Computing in Science & Engineering*, *9*(3), 90–95.

Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., … Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, *585*, 357–362. https://doi.org/10.1038/s41586-020-2649-2

McKinney, W., & others. (2010). Data structures for statistical computing in python. In *Proceedings of the 9th Python in Science Conference* (Vol. 445, pp. 51–56).